CSE2110-001,002 Fall-2021

Project 2: Airport Dashboard

Milestone Project 1 Submission Due: November 15 by 11:59PM
Milestone Project 2 Submission Due: November 22 by 11:59PM
Final Submission Due: December 5 by 11:59PM

Instructions

For this project you will be designing and implementing a system, in C or C++ to store employee data. **Your system should act as a database and allow the user to load in multiple tables of data, run basic queries on the data, and then store the data off for later use.**

Additionally, sample input files can be found on the project zip file. Further, you will need to utilize the GitLab code repository located at https://csegitlab.cse.unt.edu/. You may access it from the website, the terminal, or an IDE of your choice. Remember while accessing the repository you must use UNT VPN otherwise you may not access it.

Also, as a reminder, all the work for this project must be the sole product of the group. You may not share code with other groups, or download solutions off the internet, as doing so will be considered cheating. If you are struggling with a concept or implementation, you are encouraged to contact the instructor or your TA's for aid.

Requirements

This assignment has two parts: a wiki portion and an implementation portion.

Wiki

For the wiki portion, the group must generate documentation describing their system using the wiki functionality provided by GitLab. The wiki must contain at least one page that provides a high-level description of the entire system and must contain at least one page for each of the major functionality components (i.e. if there are four major functionality components, there should be at least four pages).

For the high-level description page, the team must provide a brief description of each of the major functionality components, how they work together, and some of the major data structures

across the entire project. This information should not be copied and pasted from the project instructions. The page must also contain a diagram of the entire system, based on the one created during recitations. The diagram must be created digitally (i.e. using PowerPoint, Photoshop, Paint.net, UMLet, etc.), must be easy to read and understand, and cannot a photographed or scanned image.

For each major functionality component page, the student accountable for that component must provide a detailed description of their component. This description should have three labeled sections: a brief description of the purpose of the component, a description of how data was stored and maintained for this component, and a description of the functionality for the component. They might also consider including diagrams to more easily visualize how all of the pieces fit together.

For the data storage and maintenance section, there should be an explanation of how data was stored and maintained in their component. What, if any, objects or structs were created to store data? How were they organized and managed? What types of formal data structures did were made use of (trees, graphs, arrays, hashes, etc)?

For the functionality component, there should be an explanation of the major functions in the component. How is data moved and transformed? How is it read in? How is it output? What are the various major functions constructed and how do they work?

Descriptions and explanations should be primarily in prose and paragraph format, not bulleted lists. Code snippets are also acceptable, but must be used as an enhancement to the explanation of functionality not as a substitution for it. Your grade for the wiki will partly be based on apparent effort, so please be thorough in your descriptions. Additionally, because this is a wiki, the high-level description page must have links to all of the major functionality component pages.

Implementation

Your program must provide the following functionality and adhere to the following constraints:

- Your int main() should be in its own .c/.cpp file
- All .c/.cpp files, except your main.c/main.cpp, must have associated header files. You may not #include a .c/.cpp file, only header files
- Allow the user to choose the file describing the initial setup of the database
    - Do **NOT** hardcode the filename into your program
    - The first set of lines in this file will provide a space delimited pair of a file name and a table name that should be used for the data within that file. There will be one pair per line, and the pairs may be in any order
    - The next line will be an empty line
    - **The remaining lines will be basic operations including <u>INSERT, UPDATE, SELECT, DELETE, DISPLAY and WRITE</u>**
- **Table's Schemas and Relations information**
    - The table's schema is given in **tables_schema.csv** file.

- In each row: the first column is the name of the tables and the rest are data elements.
    - There is colon '**:'** separator in each column. The key for each table is defined by **"k"** followed by **':'** separator**.** Some tables may have compound keys.
    - The type of data will be either string or integer which is defined as **":str"** or **":int"**.

- **The relationship of tables is given in airport.png.** This file shows the key for each table and relation with other tables.

- **Each table has an individual .csv data file:**
    a. All lines will be comma-delimited rows of data to be inserted into the table. (Comma symbol is used as the delimited columns in each row.)
    b. First row is the name of columns.
    c. Missing element in row should be considered as "NULL"
2. **While the data may change for different sets of input files, the schemes will always be identical for each table**

Milestone Project 1 Submission Items:

1. **Each table should be implemented as a hash table**
    - You must have one unique table class per table
    - A table class must be defined in its own .cpp and header
    - An entry in a table may be defined either as a class or a struct, and may be a private inner class/struct
    - You must implement your own hash table and hash function
        - Initiate table size based on longest CSV data file. (you set table size as 23)
        - You may choose your collision strategy from the strategies we have covered in class
        - You may choose your hashing function from the strategies we have covered in class
        - You have to use all hashing functions and collision strategies which are reviewed during the class.
        - The hash function must be implemented as a separate function
        - You must indicate which hash function and collision strategy your table uses via comments
    - You may not use the map, unordered_map, or any other hash table from any of the C/C++ libraries, STL, or Boost libraries but can use vector
    - You may not use the hash function or any other hashing function from any of the C/C++ libraries, STL, or Boost libraries but can use vector

2. **All .cpp files, except your main.cpp, must have associated header files. You may not #include a .cpp file, only header files**
3. **Basic Queries**

   The queries should be implemented as individual functions in the following way: (this section will be done in three phases).

   1. **DISPLAY()**
      - DISPLAY should output the current state of each table to the user in a tabular format
      - Be sure to include the attribute names for each table Queries must be processed via a regular expression

Milestone Project 2 Submission

Continue implementing basic queries from milestone 1. Your program must provide all requested functionality for major functionality components 2,3, and 4 along with milestone 1.

2. **INSERT(string, string)** where the first string is a tuple of key & values, and second string is the table name
   a) INSERT should report to the user and not overwrite the data if there is already an entry with that key value. Key should be unique. The user does not allow to add repeated key in table
   b) Otherwise INSERT should add the data to the table and report a successful insertion to the user
   c) Be sure to specify the tuple and the table when reporting to the user

3. **UPDATE(string, string)** where the first is a tuple of key & values, and second string is the table name
   a) UPDATE should report to the user and not update the data if there is no entry with that key value
   b) Otherwise UPDATE should alter the data that key points to and report a successful update to the user
   c) Be sure to specify the tuple and the table when reporting to the user
   d) Key should be unique. The user does allow to add repeated key in table

4. **SELECT(string ,string),** where the first string is a tuple that can have either a value or * for each component and string is the table name and second string is the value of column which you look for. For example **SELECT((*,*,*,*,*,*,*),Pilot),Jones)**. It returns row of table which the name is equal to Jones → (1,Jones,5/10/1990)
   a) SELECT should report to the user if no rows match the query
   b) Otherwise SELECT should report to the user all rows that match the query
   c) Be sure to specify the tuple and the table when reporting to the user

Final Project Submission

Your program must provide all requested functionality for major functionality components 5 and 6 along with milestone 2.

5. **DELETE**(**string, string)** where the first is a tuple that can have either a key & value or * for all tuples, and second string is the table name
    - DELETE should report to user if no rows match the query
    - Otherwise DELETE should remove all rows that match the query and report a successful deletion to the user
    - Be sure to specify the tuple and the table when reporting to the user
6. **WRITE()**
    - WRITE should write the data in each of the tables to a .CSV separate file in the same format they were read
    - All other lines will be comma delimited rows of data in the table
    - The filename should be the different than the original file name to avoid overwriting. (e.g "table_name_v1.csv")

- Major functionality components must be constructed in some function, or across some functions, that are declared and defined outside of your main.c/main.cpp . Remember, function declarations must be stored in a header file, while definitions must be stored in a .c/.cpp file. You may have additional functions that support your major functionality component function.
- Your code must be well commented.

- Each group member should be performing regular commits to the GitLab repository on GitLab server with meaningful commit messages. “Fixed bug” or “New code” are not meaningful, so try to be more specific about what was fixed or what was added.
- Please do not commit the example input and output files to the remote server as that may cause a space issue on the server.
- You must provide a .txt format README file which includes:
    - The names of all group members
    - Instructions on how to compile your program
    - Instructions on how to run your program

- Additionally, you may write a makefile, but please specify if it needs any additional flags to function properly
- Each student must be accountable for one or more major functionality components and may not swap after they sign up for a component barring an exceptional circumstance. Failure to be accountable for any major functionality component will result in a 0 for the coding portions of the project (milestone submission and/or final submission). Keep in mind that some components build on others, so be careful about who takes ownership of which pieces and manage your time to avoid a crunch near the due date. Also, the group should strive to balance the work across all members. The major functionality components are:
    1. Reading in the input file and organizing initial data storage, reading in queries, and output the data to the screen
    2. Creating (Plane, PlaneType, PlaneSeats) table to store the data, and developing functions to insert, update, select, delete, and write that data to file

3. Creating (FlightLeg , FlightInstance, FlightLegInsatnce) table to store the data, and developing functions to insert, update, select, delete, and write that data to file
4. Creating (Passenger, Reservation) table to store the data, and developing functions to insert, update, select, delete, and write that data to file
5. Creating (Airport, Pilot) table to store the data, and developing functions to insert, update, select, delete, and write that data to file

Milestone Project 1 Submission

Your program must provide all requested functionality for major functionality component 1. At least one group member must submit a .zip file containing the following:

- All files necessary to compile and run your program (Please do not include any files not necessary to run the program on the CSE machines)
- A .txt format README file explaining how to compile and run your program

Milestone Project 2 Submission

Your program must provide all requested functionality for major functionality components 2,3, and 4 along with milestone 1. At least one group member must submit a .zip file containing the following:

- All files necessary to compile and run your program (Please do not include any files not necessary to run the program on the CSE machines)
- A .txt format README file explaining how to compile and run your program

Final Project Submission

Your program must provide all requested functionality for major functionality components 5 and 6 along with milestone 2. At least one group member must submit a .zip file containing the following:

- All files necessary to compile and run your program (Please do not include any files not necessary to run the program on the CSE machines)
- A .txt format README file explaining how to compile and run your program

Rubric

The entire assignment is worth 300 points, and each student will receive a single grade with respect to both the entire project and to the portions they were individually responsible for. The breakdown of those points is as follows.

- 20 points: Project Expectations Document
- 30 points: Project Design
- 15 points: Project Check-ins
- 10 points: GitLab Commits
- 25 points: Project Wiki
- 100 points: Project Milestone Submission
    - 80 points: Implemented functionality
    - 20 points: Proper Commenting
- 100 points: Project Final Submission

- 80 points: Implemented functionality
- 10 points: Proper Commenting
- 10 points: Group Evaluation

If your code fails to compile on the CSE machines you will not receive credit for the code portion of the assignment (milestone submission and/or final submission). I recommend not making changes to your code without checking for compilation before you submit.